Amendments to the Claims

The listing of claims below will replace all prior versions of claims in the application:

1. (*currently amended*) A method for precise feedback data generation and updating during compile-time optimizations, within an optimizing compiler, comprising:

[[(1)]] (a) accessing a first intermediate representation of source code of a computer program, wherein said first intermediate representation includes instructions instrumented into the source code of said computer program;

[[(2)]] (b) annotating said first intermediate representation with previously-gathered frequency data from a plurality of sample executions of said computer program;

[[(3)]] (c) updating said frequency data to maintain accuracy of said frequency data during compilation in a direction of increasing exactness;

[[(4)]] (d) performing an optimization of said first intermediate representation annotated with said frequency data updated in step [[(3)]] (c), thereby producing a transformed intermediate representation; and

[[(5)]] (e) repeating steps [[(3)]] (c) and [[(4)]] (d) at least once during the same compilation pass.

2. (*currently amended*) The method of claim 1, wherein step [[(4)]] (d) comprises the step of performing at least one of the following optimizations:

(i) dead code elimination;

(ii) dead store elimination;

(iii) branch elimination; and

(iv) code transformation.

3. (*original*) The method of claim 1, wherein said first intermediate representation is a tree corresponding to a procedure within the source code of said computer program.

4. (*currently amended*) The method of claim 3, wherein step [[(2)]] (b), comprises the steps of:

[[(a)]] (i) constructing a control flow graph from said tree; and

[[(b)]] (ii) annotating a frequency value to an edge of said control flow graph, wherein said frequency value corresponds to the number of times that said edge was traversed during said plurality of sample executions of said computer program.

5. (*currently amended*) The method of claim 4, wherein said frequency value annotated to said edge of said control flow graph is one of the following:

[[(i)]] (a) EXACT;

[[(ii)]] (b) GUESS;

[[(iii)]] (c) UNKNOWN;

[[(iv)]] (d) UNINIT; and

[[(v)]] (e) ERROR.

6. (*previously presented*) A computer program product comprising a computer usable medium having computer readable program code means embodied in said medium for causing an application program to execute on a computer that performs precise feedback data generation and updating during compile-time optimizations, within an optimizing compiler, said computer readable program code means comprising:

first computer readable program code means for causing the computer to access a first intermediate representation of source code of a computer program, wherein said first intermediate representation includes instructions instrumented into the source code of said computer program;

second computer readable program code means for causing the computer to annotate said first intermediate representation with previously-gathered frequency data from a plurality of sample executions of said computer program;

third computer readable program code means for causing the computer to update said frequency data to maintain accuracy of said frequency data during compilation in a direction of increasing exactness;

fourth computer readable program code means for causing the computer to perform an optimization of said first intermediate representation annotated with said frequency data updated by said third computer readable program code means, thereby producing a transformed intermediate representation; and

fifth computer readable program code means for causing the computer to re-execute said third and fourth computer readable program code means at least once during the same compilation pass.

7. (*original*) The computer program product of claim 6, wherein said first intermediate representation is a tree corresponding to a procedure within the source code of said computer program.

8. (*original*) The computer program product of claim 7, wherein said second computer readable program code means comprises:

sixth computer readable program code means for causing the computer to construct a control flow graph from said tree; and

seventh computer readable program code means for causing the computer to annotate a frequency value to an edge of said control flow graph, wherein said frequency value corresponds to the number of times that said edge was traversed during said plurality of sample executions of said computer program.

9. (*currently amended*) A method for compile-time optimization comprising:

[[(1)]] (a) accessing a first intermediate representation of source code of a computer program, wherein the first intermediate representation includes instructions instrumented into the source code;

[[(2)]] (b) annotating the first intermediate representation with previously-gathered global and local frequency data from a plurality of sample executions of the computer program;

[[(3)]] (c) updating the global and local frequency data to maintain accuracy of said frequency data during compilation in a direction of increasing exactness;

[[(4)]] (d) performing an optimization of the first intermediate representation annotated with the global and local frequency data updated in step [[(3)]] (c) to produce a transformed intermediate representation; and

[[(5)]] (e) repeating steps [[(3)]] (c) and [[(4)]] (d) at least once during the same compilation pass.

10. (*currently amended*) The method of claim 9, wherein step [[(4)]] (d) comprises the step of performing at least one of the following optimizations:

(i) dead code elimination;

(ii) dead store elimination;

(iii) branch elimination; and

(iv) code transformation.

11. (*previously presented*) The method of claim 9, wherein the first intermediate representation is a tree corresponding to a procedure within the source code.

12. (*currently amended*) The method of claim 11, wherein step [[(2)]] (b) comprises the steps of:

[[(a)]] (i) constructing a control flow graph from the tree; and

[[(b)]] (ii) annotating a global or local frequency value of the global and local frequency data to an edge of the control flow graph, wherein the global or local frequency value corresponds to the number of times that the edge was traversed during the plurality of sample executions of the computer program.

13. (*currently amended*) The method of claim 12, wherein the global and local frequency value annotated to the edge of the control flow graph is one of the following:

[[(i)]] (a) EXACT;

[[(ii)]] (b) GUESS;

[[(iii)]] (c) UNKNOWN;

[[(iv)]] (d) UNINIT; and

[[(v)]] (e) ERROR.

14. (*currently amended*) A method for compile-time optimization comprising:

[[(1)]] (a) accessing a first intermediate representation of source code of a computer program, wherein the first intermediate representation includes instructions instrumented into the source code;

[[(2)]] (b) annotating the first intermediate representation with previously-gathered frequency data from a plurality of sample executions of the computer program;

[[(3)]] (c) updating the frequency data to maintain accuracy of said frequency data in a direction of increasing exactness at multiple points during a compilation process;

[[(4)]] (d) performing an optimization of the first intermediate representation annotated with the frequency data updated in step [[(3)]] (c) to produce a transformed intermediate representation; and

[[(5)]] (e) repeating steps [[(3)]] (c) and [[(4)]] (d) at least once during the same compilation pass.

15. (*currently amended*) The method of claim 14, wherein step [[(4)]] (d) comprises the step of performing at least one of the following optimizations:

(i) dead code elimination;

(ii) dead store elimination;

(iii) branch elimination; and

(iv) code transformation.

16. (*previously presented*) The method of claim 14, wherein the first intermediate representation is a tree corresponding to a procedure within the source code.

17. (*currently amended*) The method of claim 16, wherein step [[(2)]] (b) comprises the steps of:

[[(a)]] (i) constructing a control flow graph from the tree; and

[[(b)]] (ii) annotating a global or local frequency value of the global and local frequency data to an edge of the control flow graph, wherein the global or local frequency value corresponds to the number of times that the edge was traversed during the plurality of sample executions of the computer program.

18. (*currently amended*) The method of claim 17, wherein the global or local frequency value annotated to the edge of the control flow graph is one of the following:

[[(i)]] (a) EXACT;

[[(ii)]] (b) GUESS;

[[(iii)]] (c) UNKNOWN;

[[(iv)]] (d) UNINIT; and

[[(v)]] (e) ERROR.

19. (*currently amended*) The method of claim 17, wherein the global or local frequency value annotated to the edge of the control flow graph is one of the following:

[[(i)]] (a) GUESS; and

[[(ii)]] (b) UNKNOWN.

20. (*currently amended*) A method for compile-time optimization comprising the steps of:

[[(1)]] (a) accessing a first intermediate representation of source code of a computer program, wherein the first intermediate representation includes instructions instrumented into the source code;

[[(2)]] (b) annotating the first intermediate representation with previously-gathered estimated frequency data from a plurality of sample executions of the computer program;

[[(3)]] (c) updating the estimated frequency data to maintain accuracy of said frequency data during compilation in a direction of increasing exactness;

[[(4)]] (d) performing an optimization of the first intermediate representation annotated with the estimated frequency data updated in step [[(3)]] (c) to produce a transformed intermediate representation; and

[[(5)]] (e) repeating steps [[(3)]] (c) and [[(4)]] (d) at least once during the same compilation pass.

21. (*previously presented*) The method of claim 1, wherein said frequency data comprises both inexact and exact values.

22. (*currently amended*) The method of claim 1, wherein ~~said updating~~ step (c) updates said frequency data from GUESS to EXACT values.

23. (*currently amended*) The method of claim 1, wherein ~~said updating~~ step (c) updates said frequency data from UNKNOWN to GUESS values.

24. (*currently amended*) The method of claim 1, wherein ~~said updating~~ step (c) updates said frequency data from UNINIT to GUESS values.

25. (*previously presented*)The method of claim 1, wherein said optimization is performed in a direction of decreasing exactness.

26. (*previously presented*)The method of claim 1, wherein said optimization is performed locally, and the updating is performed globally.

27. (*previously presented*) The product of claim 6, wherein said frequency data comprises both inexact and exact values.

28. (*currently amended*) The product of claim 6, wherein said ~~updating step updates~~ said third computer readable program code means includes:

computer readable program code means for causing the computer to update said frequency data from GUESS to EXACT values.

29. (*currently amended*) The product of claim 6, wherein said ~~updating step updates~~ said third computer readable program code means includes:

computer readable program code means for causing the computer to update said frequency data from UNKNOWN to GUESS values.

30. (*currently amended*) The product of claim 6, wherein said ~~updating step updates~~ said third computer readable program code means includes:

computer readable program code means for causing the computer to update said frequency data from UNINIT to GUESS values.

31. (*previously presented*) The product of claim 6, wherein said optimization is performed in a direction of decreasing exactness.

32. *(previously presented)* The product of claim 6, wherein said optimization is performed locally, and the updating is performed globally.

33. *(previously presented)* The method of claim 9, wherein said frequency data comprises both inexact and exact values.

34. *(currently amended)* The method of claim 9, wherein ~~said updating~~ step (c) updates said frequency data from GUESS to EXACT values.

35. *(currently amended)* The method of claim 9, wherein ~~said updating~~ step (c) updates said frequency data from UNKNOWN to GUESS values.

36. *(currently amended)* The method of claim 9, wherein ~~said updating~~ step (c) updates said frequency data from UNINIT to GUESS values.

37. *(previously presented)* The method of claim 14, wherein said frequency data comprises both inexact and exact values.

38. *(previously presented)* The method of claim 9, wherein said optimization is performed in a direction of decreasing exactness.

39. *(previously presented)* The method of claim 9, wherein said optimization is performed locally, and the updating is performed globally.

40. *(currently amended)* The method of claim 14, wherein ~~said updating~~ step (c) updates said frequency data from GUESS to EXACT values.

41. (*currently amended*) The method of claim 14, wherein ~~said updating~~ step (c) updates said frequency data from UNKNOWN to GUESS values.

42. (*currently amended*) The method of claim 14, wherein ~~said updating~~ step (c) updates said frequency data from UNINIT to GUESS values.

43. (*previously presented*) The method of claim 14, wherein said optimization is performed in a direction of decreasing exactness.

44. (*previously presented*) The method of claim 14, wherein said optimization is performed locally, and the updating is performed globally.

45. (*new*) A method for precise feedback data generation and updating during compile-time optimizations, within an optimizing compiler, comprising:

(a) accessing an intermediate representation of source code of a computer program, wherein the intermediate representation includes instructions instrumented into the source code;

(b) annotating the intermediate representation with frequency data from a plurality of executions of the computer program;

(c) performing an optimization to produce a transformed intermediate representation;

(d) re-annotating the transformed intermediate representation, wherein step (d) comprises:

(i) constructing a control flow graph, wherein said control flow graph includes a plurality of nodes representing portions of the transformed intermediate representation, wherein said control flow graph further includes a plurality of edges, each edge representing a relationship between one of the nodes to another node;

(ii) applying available frequency data to an edge to set an incoming edge frequency or an outgoing edge frequency for a node;

(iii) propagating a known edge frequency to set an unknown edge frequency for a node; and

(iv) re-annotating the transformed intermediate representation with the edge frequencies from the control flow graph; and

(v) repeating steps (c) and (d), at least once during the same compilation pass, wherein step (c) includes performing an optimization of the re-annotated transformed intermediate representation from step (d) to produce a transformed intermediate representation for re-annotating at a subsequent execution of step (d).

46. (*new*) The method of claim 45, wherein step (d)(iii) comprises:

(a) setting an unknown edge frequency for an outgoing edge to a difference between a sum of all known incoming edge frequencies and a sum of all known outgoing edge frequencies when all incoming edge frequencies are known and all but one of the outgoing edge frequencies are known;

(b) setting an unknown edge frequency for an incoming edge to a difference between a sum of all known outgoing edge frequencies and a sum of all known incoming edge frequencies when all outgoing edge frequencies are known and all but one of the incoming edge frequencies are known;

(c) setting an unknown edge frequency for an outgoing edge to zero when all incoming edge frequencies are exact values and the sum of all incoming edge frequencies equals the sum of all outgoing edge frequencies; and

(d) setting an unknown edge frequency for an incoming edge to zero when all outgoing edge frequencies are exact values and the sum of all outgoing edge frequencies equals the sum of all incoming frequencies.

47. (*new*) The method of claim 45, wherein step (d)(iii) comprises:

denoting a node as being a non-exception node when the total incoming edge frequency and the total outgoing edge frequency for the node are required to be equal.

48. (*new*) The method of claim 47, wherein step (d)(iii) is only executed on the non-exception node.

49. (*new*) A method for precise feedback data generation and updating during compile-time optimizations, within an optimizing compiler, comprising:

(a) accessing an intermediate representation of source code of a computer program, wherein the intermediate representation includes instructions instrumented into the source code;

(b) annotating the intermediate representation with frequency data from a plurality of executions of the computer program;

(c) performing an optimization to produce a transformed intermediate representation;

(d) re-annotating and verifying the re-annotation of the transformed intermediate representation; and

(e) repeating steps (c) and (d), at least once during the same compilation pass, wherein step (c) includes performing an optimization of the re-annotated transformed intermediate representation from step (d) to produce a transformed intermediate representation for re-annotating at a subsequent execution of step (d).

50. (*new*) The method of claim 49, wherein step (d) comprises:

evaluating a portion of the transformed intermediate representation to classify a corresponding frequency data annotation as being uninitialized or invalid.

51. (*new*) The method of claim 49, wherein step (d) comprises:

evaluating a portion of the transformed intermediate representation to classify a corresponding frequency data annotation as being exact, estimated, or unknown.